212CSE2303- Software Engineering

UNIT I: Software Engineering Concepts

Software and Software Engineering - Project Management Concepts - Software Engineering Paradigms Generic Process Models, Assessment and Improvement, Use Case Model: Goals, Actors - Finding Primary Actors, Use Case types and Formats.

Software and Software Engineering:

Software:

Software is a program or set of programs containing instructions that provide desired functionality.

Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

The Nature of Software

Software takes Dual role of Software. It is a Product and at the same time a Vehicle for delivering a product.

Software delivers the most important product of our time is called **information**

Defining Software

Software is defined as

1. Instructions : Programs that when executed provide desired function, features, and performance

2. Data structures : Enable the programs to adequately manipulate information

3. Documents: Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

Characteristics of Software:

1. It is intangible, ➔meaning it cannot be seen or touched.
2. It is non-perishable, ➔ meaning it does not degrade over time.
3. It is easy to replicate, ➔ meaning it can be copied and distributed easily.
4. It can be complex, ➔ meaning it can have many interrelated parts and features.
5. It can be difficult to understand and modify, ➔ especially for large and complex systems.
6. It can be affected by changing requirements, ➔ meaning it may need to be updated or modified as the needs of users change.
7. It can be affected by bugs and other issues, ➔ meaning it may need to be tested and debugged to ensure it works as intended.

Software Applications:

1. **System Software:**

   System Software is necessary to manage the computer resources and support the execution of application programs.

   Software like operating systems, compilers, editors and drivers, etc., come under this category.

   A computer cannot function without the presence of these. Operating systems are needed to link the machine-dependent needs of a program with the capabilities of the machine on which it runs. Compilers translate programs from high-level language to machine language.

2. **Application Software:**

   Application software is designed to fulfill the user's requirement by interacting with the user directly. It could be classified into two major categories:- **generic or customized**.

   1. Generic Software is the software that is open to all and behaves the same for all of its users. Its function is limited and not customized as per the user's changing requirements.

   2. Customized software is the software products that are designed as per the client's requirement, and are not available for all.

3. **Networking and Web Application Software**:

   Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities.

   Networking software is also used when software is running on a network of computers (such as the World Wide Web).

   It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.

4. **EmbeddedSoftware:**

   This type of software is embedded into the hardware normally in the Read-Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions.

   Examples are software used in instrumentation and control applications like washing machines, satellites, microwaves, etc.

5. **Reservation Software:**

   A Reservation system is primarily used to store and retrieve information and perform transactions related to air travel, car rental, hotels, or other activities.

They also provide access to bus and railway reservations. These are also used to relay computerized information for users in the hotel industry, making a reservation and ensuring that the hotel is not overbooked.

6. **Business Software:**
   This category of software is used to support business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospitals, schools, stock markets, etc.

7. **Entertainment Software:**
   Education and entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

8. **Artificial Intelligence Software:**
   Software like expert systems, decision support systems, pattern recognition software, artificial neural networks, etc. come under this category. They involve complex problems which are not affected by complex computations using non-numerical algorithms.

9. **Scientific Software:**
   Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software is written for specific applications using principles, techniques, and formulae particular to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

10. **Utilities Software**:
    The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are anti-virus software, voice recognition software, compression programs, etc.

11. **Document Management Software:**
    Document Management Software is used to track, manage and store documents in order to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking).

On the basis of copyright:

1. **Commercial –**
   It represents the majority of software that we purchase from software companies, commercial computer stores, etc. In this case, when a user buys software, they acquire a license key to use it. Users are not allowed to make copies of the software. The copyright of the program is owned by the company.

2. **Shareware –**
   Shareware software is also covered under copyright but the purchasers are allowed to make and distribute copies with the condition that after testing the software, if the purchaser adopts it for use, then they must pay for it.
   In both of the above types of software, changes to the software are not allowed.
3. **Freeware** –
   In general, according to freeware software licenses, copies of the software can be made both for archival and distribution purposes but here, distribution cannot be for making a profit. Derivative works and modifications to the software are allowed and encouraged. Decompiling of the program code is also allowed without the explicit permission of the copyright holder.
4. **Public Domain** –
   In the case of public domain software, the original copyright holder explicitly relinquishes all rights to the software. Hence software copies can be made both for archival and distribution purposes with no restrictions on distribution. Modifications to the software and reverse engineering are also allowed.

Why Software Engineering? Software Crisis & its Solution:

Software Crisis:

- It was in the late 1960s when many software projects failed.
- Much software became over budget. Output was unreliable software which is expensive to maintain.
- Larger software was difficult and quite expensive to maintain.
- Lots of software not able to satisfy the growing requirements of the customer.
- Complexities of software projects increased whenever its hardware capability increased.
- Demand for new software increased faster compared with the ability to generate new software.

All the above issues lead to ‘Software Crisis.’

The Solution:

Solution was to the problem was transforming unorganized coding effort into software engineering discipline. These engineering models helped companies to streamline operations and deliver software meeting customer requirements.

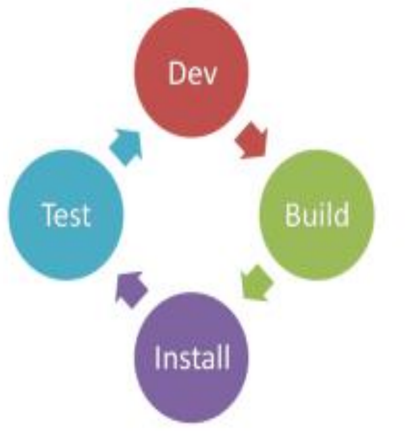
- The late **1970s** saw the widespread uses of **software engineering principles**.
- In the **1980s** saw the automation of **software engineering process and growth of (CASE) Computer-Aided Software Engineering.**

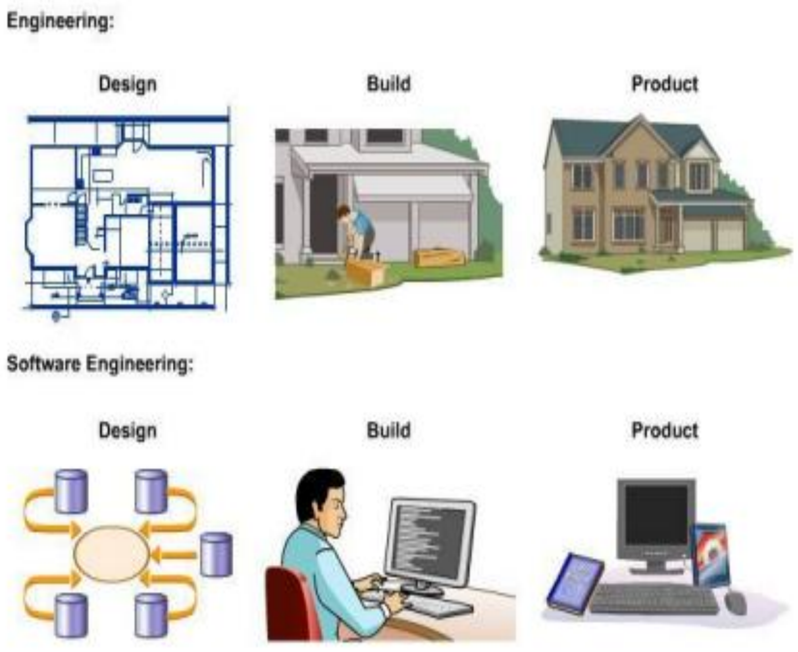
- The **1990s** have seen an increased emphasis on the 'management' aspects of **projects standard of quality and processes** just like ISO 9001

Software engineering:

Software engineering is the **process of designing, developing, testing, and maintaining software.** It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

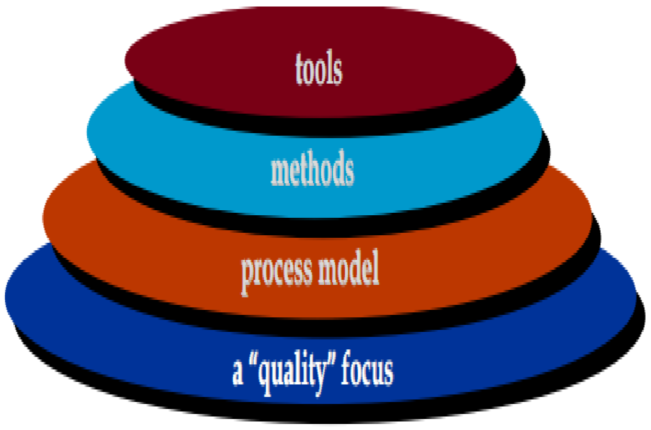
Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.





Software Engineering - A Layered Technology

Software engineering encompasses a process, the management of activities, technical methods, and use of tools to develop software products



The foundation for software engineering is the **process layer**. Software engineering process is the glue that **holds the technology layers together and enables rational and timely development of computer software**. Process defines a framework for a set of *key process areas.*

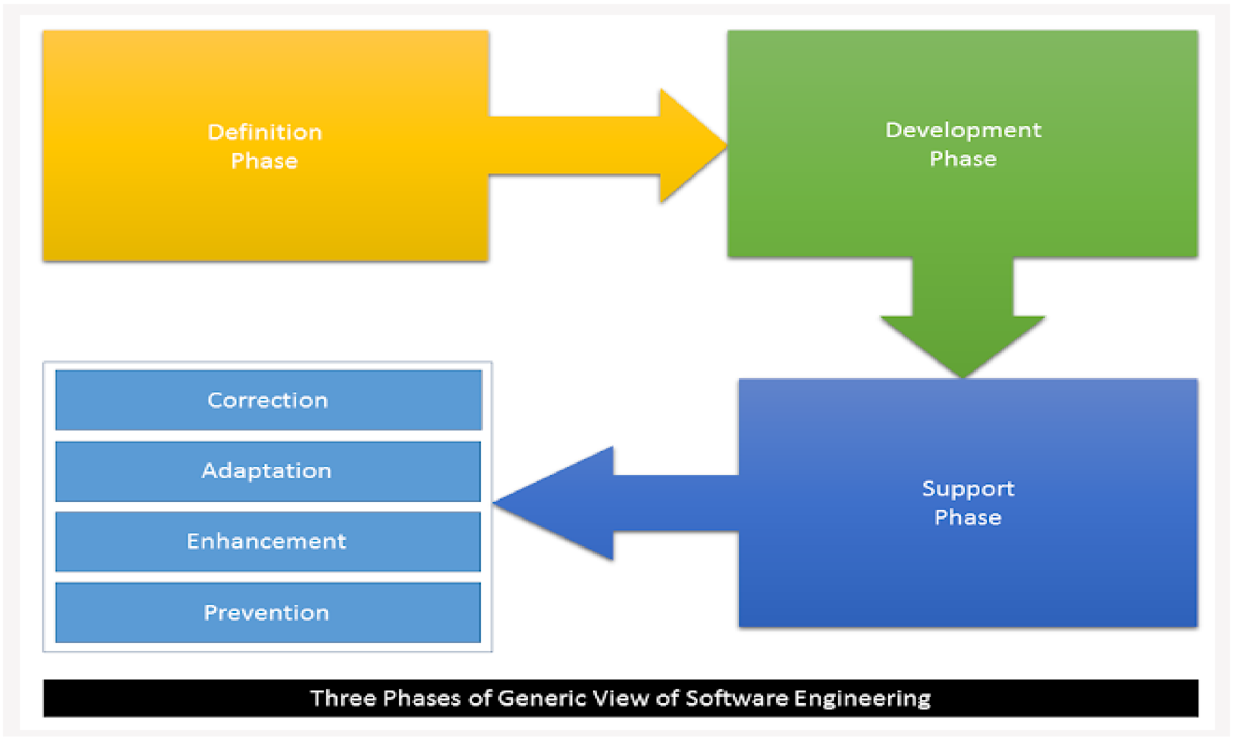
Software engineering **methods** provide the **technical how-to's for building software.** Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.

Software engineering **tools** provide **automated or semi-automated support for the process and the methods**. When tools are integrated so that information created by one tool can be used by another.

Different Phases of process:

A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)

The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity. Following flowchart encompasses the phases.



01. Definition Phase:

The definition phase focuses on **"what".** That is, during definition, the software engineer attempts to identify

1. what information is to be processed,
2. what function and performance are desired,
3. what system behavior can be expected,
4. what interfaces are to be established,
5. what design constraints exist, and
6. what validation criteria are required to define a successful system.

During this, three major tasks will occur in some form: **system or information engineering, software project planning and requirements analysis**.

02. Development Phase:

The development phase focuses on **"how".**
That is, during development a software engineer attempts to define

1.how data are to be structured,
2.how function is to be implemented within a software architecture,
3.how interfaces are to be characterized,
4.how the design will be translated into a programming language, and
5.how testing will be performed.

During this, three specific technical tasks should always occur; **software design, code generation, and software testing.**

03. Support Phase:

The support phase focuses on **“change”** associated with error correction, adaptations required as the software’s environment evolves, and changes due to enhancements brought about by changing customer requirements.

Four types of change are encountered during the support phase:

03.01 Correction:

Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.

03.02 Adaptation:

Over time, the original environment, that is, CPU, operating system, business rules etc for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

03.03 Enhancement:

As software is used, the customer/user will recognize additional functions that will provide benefit. Perfectible maintenance extends the software beyond its original functional requirements.

03.04 Prevention:

Computer software deteriorates due to change, and because of this, preventive maintenance, often called software re-engineering, and must be conducted to enable the software to serve the needs of its end users.

PROJECT MANAGEMENT CONCEPTS

Management Spectrum:

Effective project management focuses on four aspects of the project known as the 4 P's:

People- The most important element of a successful project. Key areas for software people - recruiting, selection, performance management, training, compensation, career development, organization & work design, team/culture development.

Product- The software to be built (product objectives, scope, alternative solutions, constraint)

Process - The set of framework activities and software engineering tasks to get the job done (framework activities populated with tasks, milestones, work products, and QA points)

Project - All work required to make the product a reality. (Planning, monitoring, controlling)

Player of the project

1. The Stakeholders
2. Team leaders
3. The Software Team
4. Coordination and Communication Issues.

1. **Stakeholders**
    - Senior managers who define the business issues that often have significant influence on the project.
    - Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
    - Practitioners who deliver the technical skills that are necessary to engineer a product or application.
    - Customers who specify the requirements for the software to be engineered
    - End-users who interact with the software once it is released for production use.

2. **Team Leaders**

    MOI model for leadership
    - Motivation: The ability to encourage (by "push or pull") technical people to produce to their best ability.
    - Organization: The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
    - Ideas or Innovation: The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

    Characteristics of effective project managers (problem solving, managerial identity, achievement, influence and team building)

3. **Software Teams**

    The following factors must be considered when selecting a software project team structure:
    - The difficulty of the problem to be solved
    - The size of the resultant program(s) in lines of code or function points
    - The time that the team will stay together (team lifetime)
    - The degree to which the problem can be modularized
    - The required quality and reliability of the system to be built
    - The rigidity (hardness) of the delivery date
    - The degree of sociability (communication) required for the project

"Organizational Paradigms" for software engineering team

a. Closed paradigm

Structures a team along a traditional hierarchy of authority. Such teams can work well when producing software that is quite similar to past efforts. So, Less likely to be innovative when working within the closed paradigm.

b. Random paradigm

Structures a team loosely and depends on individual initiative of the team members. When innovation or technological breakthrough is required, teams following the random paradigm will excel. It struggles when "orderly performance" is required.

c. Open paradigm

Structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm. Open paradigm team structures are well suited to the solution of complex problems but may not perform as efficiently as other teams.

d. Synchronous paradigm

Relies on the natural categorization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves.

4. **Team Coordination & Communication**
   - Characteristics of modern software
   - Scale, - development effort is high
   - Uncertainty, - continuing stream of changes
   - Interoperability - New software must communicate with existing software
     Software engineering team must establish effective methods for coordinating the people who do the work. To accomplish this, mechanisms for formal and informal communication among team members and between multiple teams must be established. Formal communication is accomplished through "writing, structured meetings, and other relatively activities. Informal communication is more personal. Members of a software team share ideas on an ad hoc basis, ask for help as problems arise, and interact with one another on a daily basis.

The Product

The product is consists of two things:

- Software or product scope
- Problem decomposition.

Product Scope

Software Scope: Scope is defined by answering the following questions:

- **Context**
  How the software to be built does fit into a larger system, product, or business context.
  What constraints are compulsory as a result of the context?
- **Information objectives**
  What customer-visible data objects are produced as output from the software?
  What data objects are required for input?
- **Function and performance**
  What function does the software perform to transform input data into output?
  Are any special performance characteristics to be addressed?
  Software project scope must be unambiguous and understandable at the management and technical levels.

Process decomposition

Process decomposition begins when the project manager tries to determine how to accomplish each activity.
E.g. A small, relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet the customer to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

The Process

The definition, development and support are the major aspects of the software process. Process model chosen must be appropriate for the:

- the customers who have requested the product and the people who will do the work,
- Characteristics of the product itself, and
- Project development environment

Once a process framework has been established, then determine the degree of thoroughness required and define a task set for each software engineering activity.

Task set =

- Software engineering tasks
- Work products
- Quality assurance points
- Milestones

THE W5HH PRINCIPLE

Barry Boehm gave a philosophy that prepares easy and manageable designs or outlines for software projects. Then he named it as **W5HH principle.**

The W5HH principle in software management exists to help project managers guide objectives, timelines, responsibilities, management styles, and resources.

W5HH questions :

1. **Why the system is going to be developed?**
   For the purpose of software work, all stakeholders must assess the validity of the system product/project. Here Barry questions that whether the project's purpose will justify the cost, time spent on it by people?
2. **What is activities are needed to be done in this?**
   In this Barry questions what task is needed to be done for a project currently.
3. **When is this done?**
   Project Scheduling is done by the team after recognizing when project tasks will be started and when they enter into the final stage to reach the goal.
4. **Who are the reasons for these activities in this project?**
   Every member who is part of the software team is responsible for this. And their roles are defined.
5. **Where are these authoritatively located?**
   Not only do software practitioners have roles in this but also users, customers, stakeholders also have roles and responsibilities organizationally.

6. **How is the job technically and managerially finished?**
   All technical strategies, management rules of the project are defined after knowing the scope of the project which is being built.
7. **How much part of each resource is required?**
   This is known by software developers after the estimation of each resource as per the needs of customers/users.

W5HH	**The Question**	**What It Means**
Why?	Why is the system being developed?	This focuses a team on the business reasons for developing the software.
What?	What will be done?	This is the guiding principle in determining the tasks that need to be completed.
When?	When will it be completed?	This includes important milestones and the timeline for the project.
Who?	Who is responsible for each function?	This is where you determine which team member takes on which responsibilities. You may also identify external stakeholders with a claim in the project.
Where?	Where are they organizationally located?	This step gives you time to determine what other stakeholders have a role in the project and where they are found.
How?	How will the job be done technically and managerially?	In this step, a strategy for developing the software and managing the project is concluded upon.
How Much?	How much of each resource is needed?	The goal of this step is to figure out the number of resources necessary to complete the project.

The Software Process Models:

Software Process:

Software Processes is a **coherent set of activities for specifying, designing, implementing and testing software systems.**

A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

- **Specification** – defining what the system should do;
- **Design and implementation** – defining the organization of the system and implementing the system;
- **Validation** – checking that it does what the customer wants;
- **Evolution** – changing the system in response to changing customer needs.

A Generic Process Model

A **process** is a collection of activities, actions, and tasks that are performed when some work product is to be created.

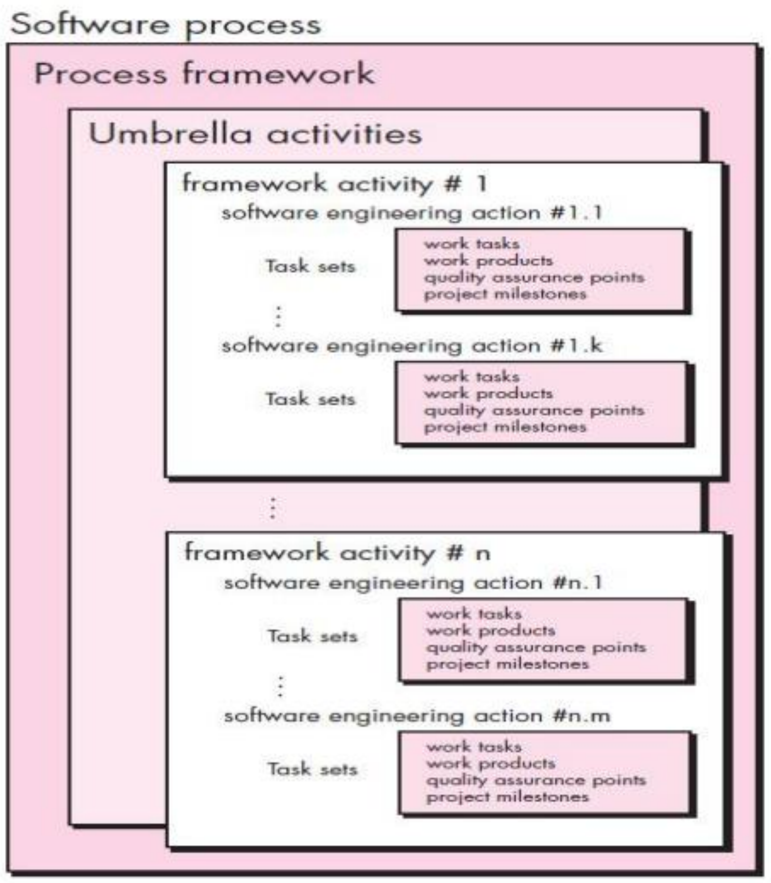
An **activity** tries to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

An **action** includes a set of tasks that produce a major work product (e.g., an architectural design model).

A **task** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

A **process framework** establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

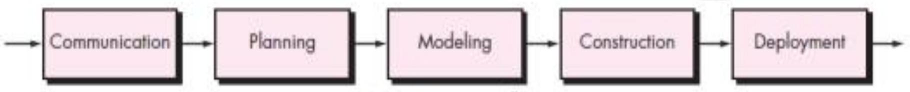
Each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.



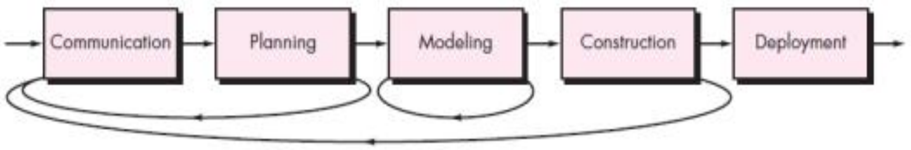
In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.

A generic process framework for software engineering encompasses **five activities:**

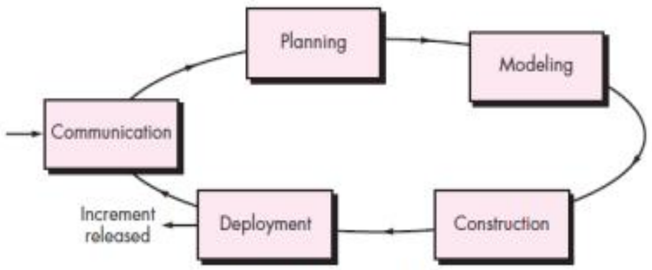
1. **Communication.** Before any technical work can commence, it is critically important to communicate and collaborate with the customer. The intent is to understand stakeholders objectives for the project and to gather requirements that help define software features and functions.
2. **Planning.** Any complicated journey can be simplified if a map exists. A software project is a complicated journey, and the planning activity creates a "map" that helps guide the team as it makes the journey. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
3. **Modeling.** Creation of models to help developers and customers understand the requires and software design.



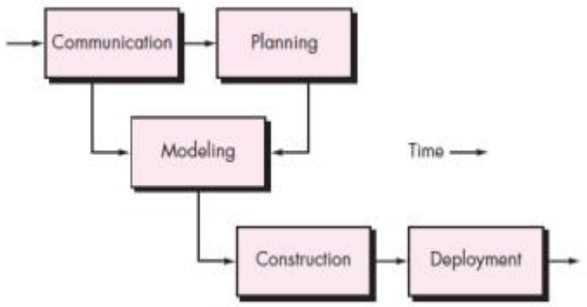
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

4. **Construction.** This activity combines code generation and the testing that is required to uncover errors in the code.
5. **Deployment**. The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the **development of small, simple programs, the creation of large Web applications**, and for the **engineering of large, complex computer-based systems.**

Software engineering process framework activities are complemented by a number of **Umbrella Activities.** In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

1. **Software project tracking and control**: allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
2. **Risk management:** assesses risks that may affect the outcome of the project or the quality of the product.
3. **Software quality assurance:** defines and conducts the activities required to ensure quality of the product.
4. **Technical reviews:** assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
5. **Measurement:** defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders needs; can be used in conjunction with all other framework and umbrella activities.
6. **Software configuration management:** manages the effects of change throughout the software process.
7. **Reusability management:** defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
8. **Work product preparation and production:** encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Advantages:

1. **Improved quality:** By following established software engineering principles and techniques, software can be developed with fewer bugs and higher reliability.
2. **Increased productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.

3. **Better maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
4. **Reduced costs:** By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.
5. **Increased customer satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
6. **Better team collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
7. **Better scalability:** By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.
8. **Better Security:** By following the software development life cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

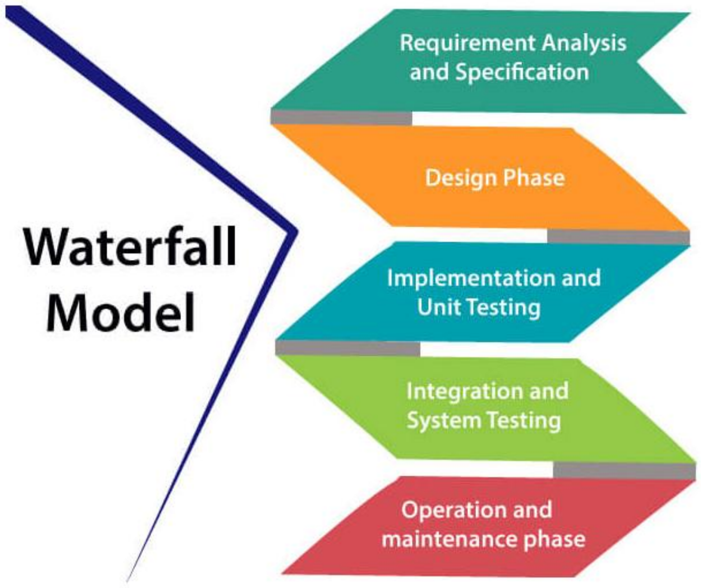
Disadvantages:

1. **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
2. **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
3. **Bureaucratic:** Software engineering can create an environment that is bureaucratic, with a lot of process and paperwork, which may slow down the development process.
4. **Complexity:** With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
5. **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
6. **High learning curve:** The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.
7. **High dependence on tools:** Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.
8. **High maintenance:** The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

Waterfall Model

Winston Royce introduced the Waterfall Model in 1970. The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.


Waterfall Model
Requirement Analysis and Specification
Design Phase
Implementation and Unit Testing
Integration and System Testing
Operation and maintenance phase

1. **Requirements analysis and specification phase:** This phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the **"what"** of the system to be produced and not "how." In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.

2. **Design Phase:** This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).
3. **Implementation and unit testing:** During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

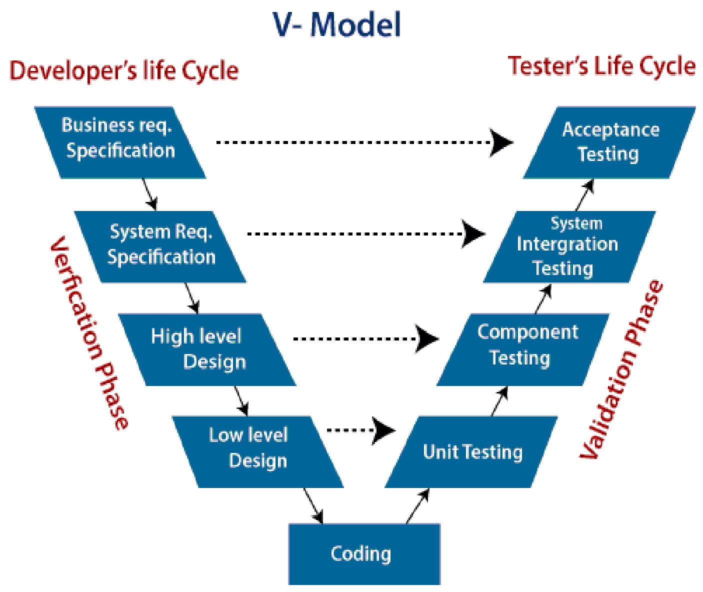
4. **Integration and System Testing:** This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.
5. **Operation and maintenance phase**: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

When to use Waterfall Model?

- ✓ When the requirements are constant and not changed regularly.
- ✓ A project is short
- ✓ The situation is calm
- ✓ Where the tools and technology used is consistent and is not changing
- ✓ When resources are well prepared and are available to use.

V Model

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.



Verification: It involves a **static analysis method (review) done** without executing code. It is the process of **evaluation of the product development process** to find whether specified requirements meet.

Validation: It involves **dynamic analysis method (functional, non-functional),** testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

Verification Phase of V-model:

1. **Business requirement analysis:** This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided.

Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.

3. **System Testing:** System Tests Plans are developed during System Design Phase. System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. Acceptance Testing: Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere.

When to use V-Model

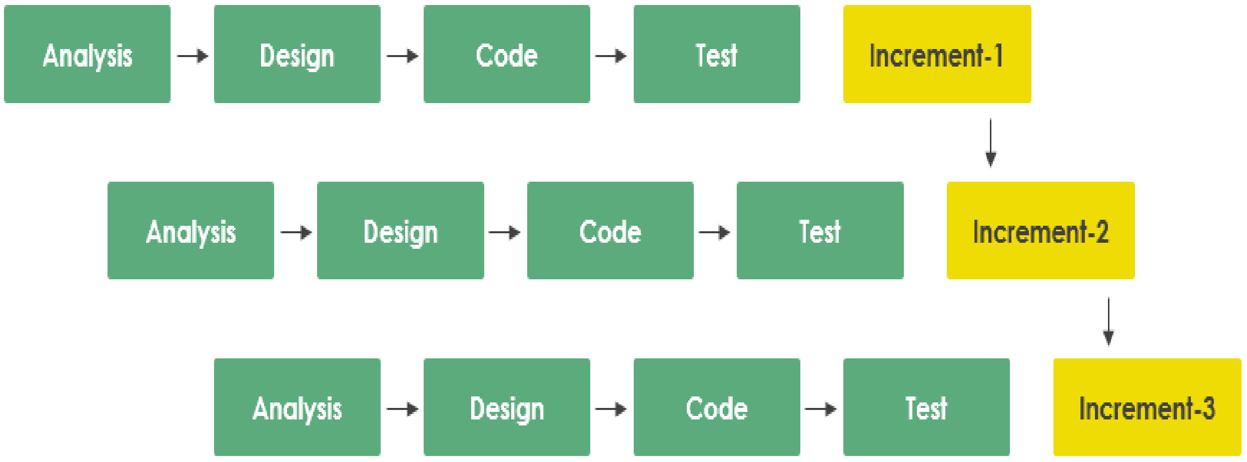
- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Incremental model

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.

It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements.

Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.



Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: The product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: The design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase.

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Iterative Model

You can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. **Every release of the Iterative Model finishes in an exact and fixed period** that is called iteration.

The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

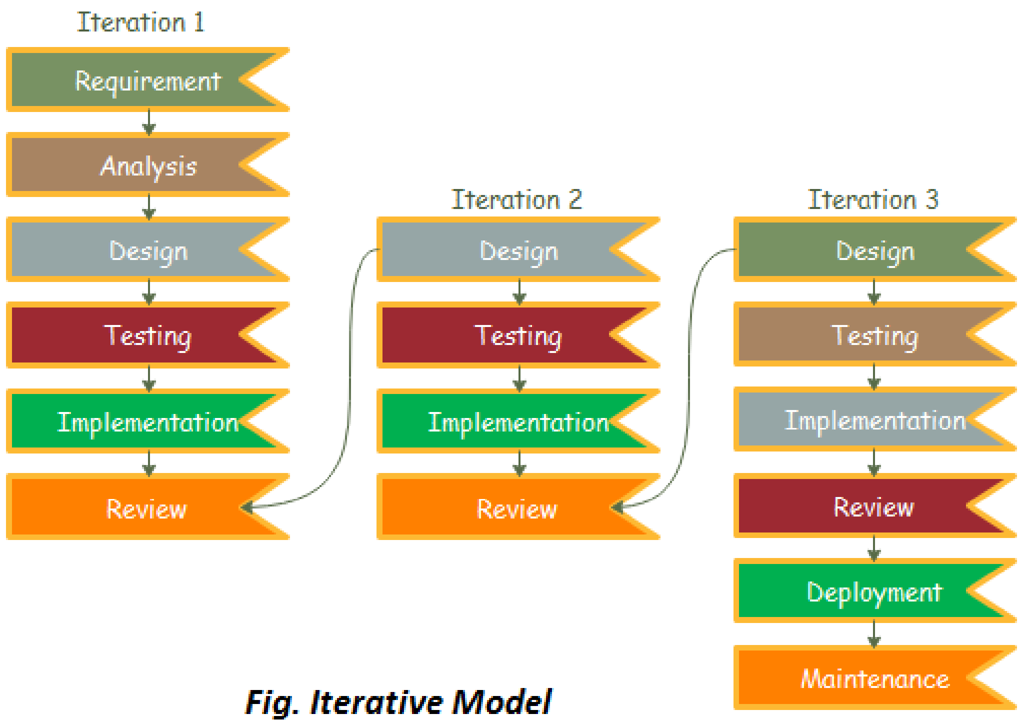


Fig. Iterative Model

Iterative model are as follows:

1. Requirement gathering & analysis: In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not.

2. Design: In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

3. Implementation: In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

4. Testing: After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

5. Deployment: After completing all the phases, software is deployed to its work environment.

6. Review: In this phase, after the product deployment, review phase is performed to check the behavior and validity of the developed product. And if there are any errors found then the process starts again from the requirement gathering.

7. Maintenance: In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

When to use the Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. When there is a requirement of changes in future.

Process assessment and improvement

Assessment attempts to understand the current state of the software process with the intent of improving it.

Software processes are assessed to ensure their ability to control the cost, time and quality of software. Assessment is done to improve the software process followed by an organization.

Software Process Improvement (SPI) Cycle includes:

- Process measurement
- Process analysis
- Process change

Approaches towards process assessment:

1.CMM (Capability Maturity Model) and CMMI (Capability Maturity Model Integration)

CMM was developed by SEI (Software Engineering Institute) and evolved into CMMI later. It is an approach based on which an organization's process maturity is determined.

CMM's five maturity levels:

Initial Level: Processes are not organized and the success of a project depends only on the competence of the individual working on it. May not be able to repeat past successes in future projects. The probability of exceeding the estimated cost and schedule is high.

Repeatable Level: In this level, successes of the past could be repeated because the organization uses project management techniques to track cost and schedule. Management according to a documented plan helps in the improved process.

Defined Level: Organization's set of standard processes are defined and are slightly modified to incorporate each project demands. This provides consistency throughout the works of the organization.

Managed Level: Management of processes using quantitative techniques improves performance. Processes are assessed through data collection and analysis.

Optimizing Level: Processes are monitored and improved through feedback from current work. Innovative techniques are applied to cope with changing business objectives and the environment.

CMMI maturity levels include:

- Initial.
- Managed.
- Defined.
- Quantitatively Managed.
- Optimized.

CMMI capability levels include:

Level 0: Incomplete – Incomplete processes are processes that are not performed or partially performed.

Level 1: Performed – Specific goals are satisfied by processes and yet certain objectives related to quality, cost and schedule are not met. Useful work can be done.

Level 2: Managed – Cost, quality and schedule are managed and processes are monitored by management techniques.

Level 3: Defined – It includes management and additionally follow the organization's specified set of standard processes which are altered for each project.

Level 4: Quantitatively Managed – Statistical and quantitative techniques are used for the management of processes.

Level 5: Optimized – It focuses on continuous improvement of Quantitatively Managed process through innovations and nature of processes.

2.Standard CMMI Appraisal Method for Process Improvement (SCAMPI)

It is a method used by Software Engineering Institute (SEI) for providing quality ratings with respect to Capability Maturity Model Integration (CMMI). Assessment includes five phases initiating, diagnosing, establishing, acting and learning. The appraisal process includes preparation, on-site activities, findings and ratings, final reporting etc.

3.CMM Based Appraisal for Internal Process Improvement (CBA IPI)

It is an SEI CMM(Capability Maturity Model) based assessment method that provides diagnostics, enables and encourages an organization to understand its maturity. It gives the organization an insight into its software development capability by assessing the strength and weakness of the current process.

4.SPICE(ISO/IEC15504)

This standard is one of the joint mission of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). They assist organizations in developing an objective evaluation of the effectiveness of a software process and related business management functions.

It consists of six levels as:

- Not performed.
- Performed informally.
- Planned and tracked.

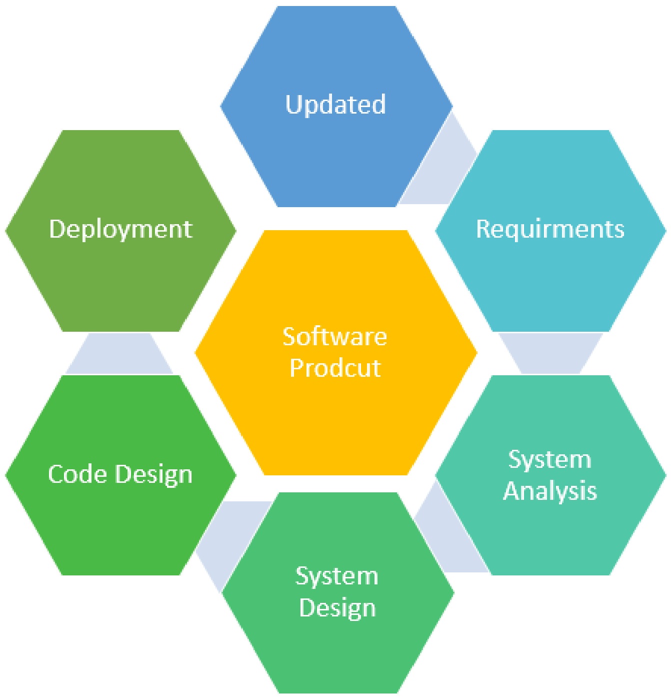
- Well defined.
- Quantitatively controlled.
- Continuously improved.

Use Case Model: Goals, Actors - Finding Primary Actors, Use Case types and Formats.

Refer -UseCase.docx

Extra Topics for referral

Why Software Engineering is Popular?



- ✓ **Large software ➔** In our real life, it is quite more comfortable to build a wall than a house or building. In the same manner, as the size of the software becomes large, software engineering helps you to build software.
- ✓ **Scalability➔** it is easier to re-create new software to scale an existing one.
- ✓ **Adaptability➔** it is easy to re-create new software with the help of software engineering.
- ✓ **Cost➔** Hardware industry has shown its skills and huge manufacturing has lower the cost of the computer and electronic hardware.
- ✓ **Dynamic Nature➔** Always growing and adapting nature of the software. It depends on the environment in which the user works.
- ✓ **Quality Management➔** Offers better method of software development to provide quality software products.

Software Process and Software Development Lifecycle Model

One of the basic notions of the software development process is SDLC models which stand for **Software Development Life Cycle models.** [

The most used, popular and important SDLC models are given below:

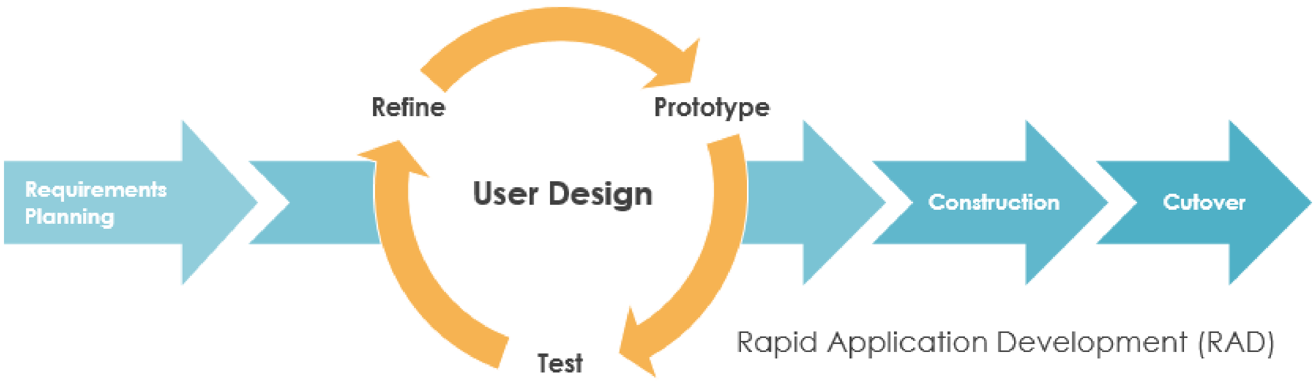
- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Spiral model
- Prototype model

RAD model

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM).

Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

RAD may resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.

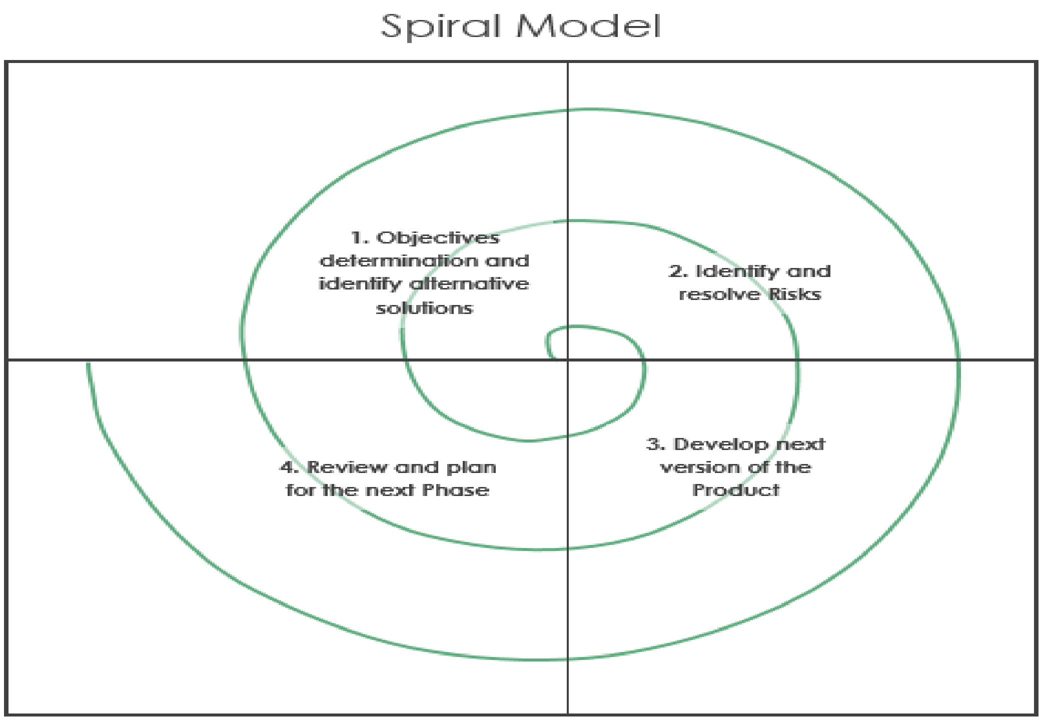


Spiral model

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model.

A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.


Spiral Model
1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Agile model

The meaning of Agile is swift or versatile."Agile process model" refers to a software development approach based on iterative development.

Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.

Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

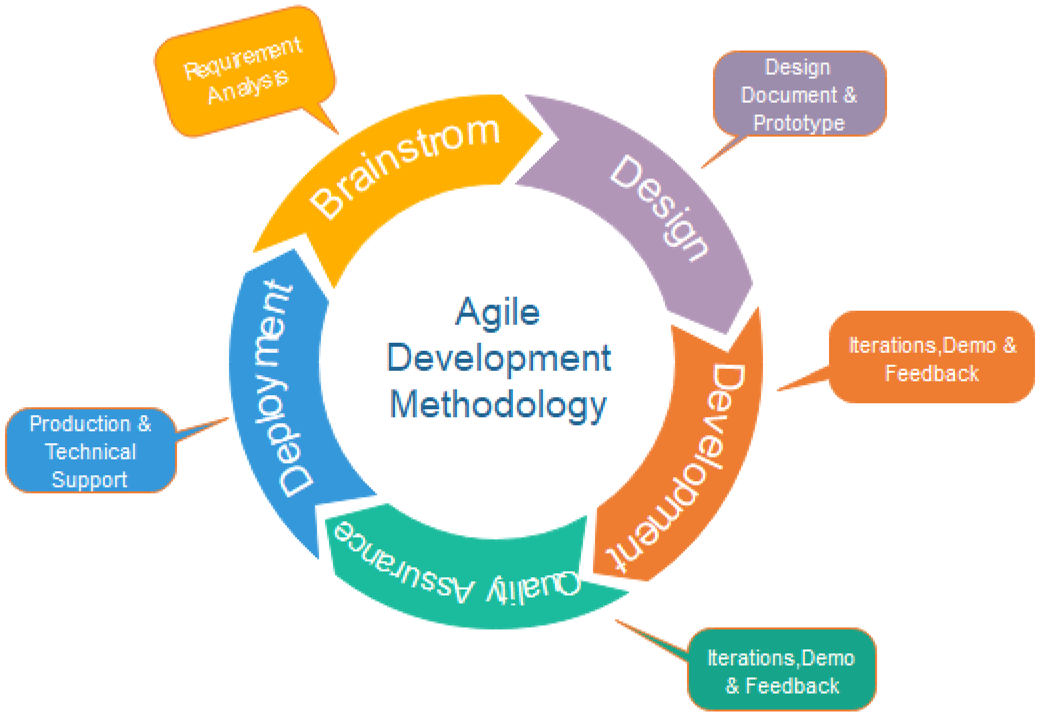

Requirement Analysis
Design Document & Prototype
Brainstrom
Design
Agile Development Methodology
Iterations,Demo & Feedback
Development
Production & Technical Support
Deployment
Quality Assurance
Iterations,Demo & Feedback

Fig. Agile Model

Objectives of Software Engineering:

1. Maintainability:
   It should be feasible for the software to evolve to meet changing requirements.
2. Efficiency:
   The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. Correctness:
   A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
4. Reusability:
   A software product has good reusability if the different modules of the product can easily be reused to develop new products.

5. Testability:
   Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
6. Reliability:
   It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. Portability:
   In this case, the software can be transferred from one computer system or environment to another.
8. Adaptability:
   In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. Interoperability: Capability of 2 or more functional units to process data cooperatively.

Key principles of software engineering include:

1. **Modularity:** Breaking the software into smaller, reusable components that can be developed and tested independently.
2. **Abstraction:** Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
3. **Encapsulation:** Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
4. **Reusability:** Creating components that can be used in multiple projects, which can save time and resources.
5. **Maintenance:** Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
6. **Testing:** Verifying that the software meets its requirements and is free of bugs.
7. **Design Patterns:** Solving recurring problems in software design by providing templates for solving them.
8. **Agile methodologies:** Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
9. **Continuous Integration & Deployment**: Continuously integrating the code changes and deploying them into the production environment.